U-Boot ACPI support on ARM64

Implementing the ARM SBBR

Patrick Rudolph <patrick.rudolph@9elements.com>

9ELEMENTS

Motivation

Task: *Enable arm SBSA on Raspberry Pi 4 within U-Boot*

Motivation

Task: *Enable arm SBSA on Raspberry Pi 4 within U-Boot*

ARM SBSA

- https://developer.arm.com/documentation/den0029/latest/

  “Server Base System Architecture (SBSA) specifies a hardware system architecture for servers that are based on the Arm 64-bit Architecture”

ARM BBR

- https://developer.arm.com/documentation/den0044/latest/

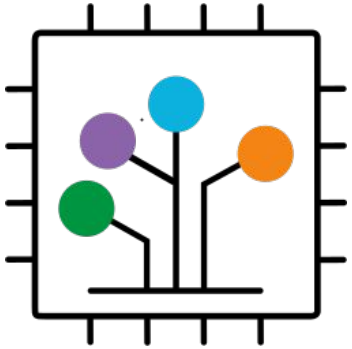
  “Base Boot Requirements (BBR) for Boot and Runtime Services”

Motivation - BBR

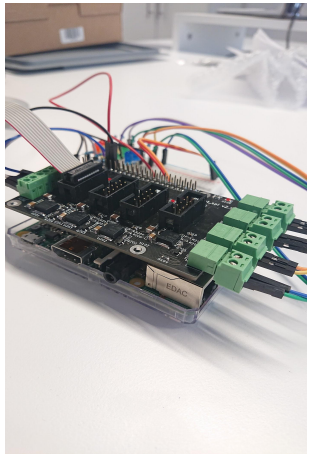
BBR mandates that are implemented in U-Boot on Aarch64:

- **UEFI** ✅
- **PSCI/SMCCC** ✅
- **ACPI 6.5 with reduced HW model** ❌
- **Full 64-bit support within ACPI** ❌
- **SMBIOS** ❌

Devicetree

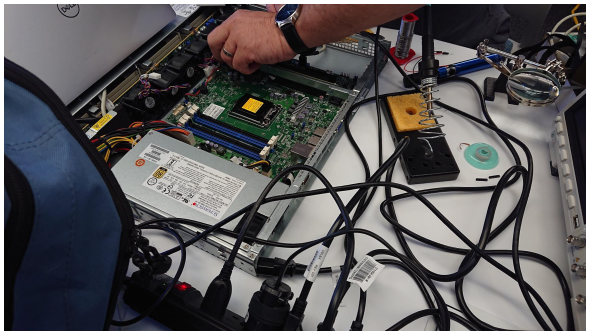


- Data structure describing the hardware
- Derived from SPARC based computers
- Adopted for embedded platforms
- Typically *not* used on x86 or server platforms
- Lossless conversion between source and binary form
- Requires a devicetree parser on the OS/bootloader



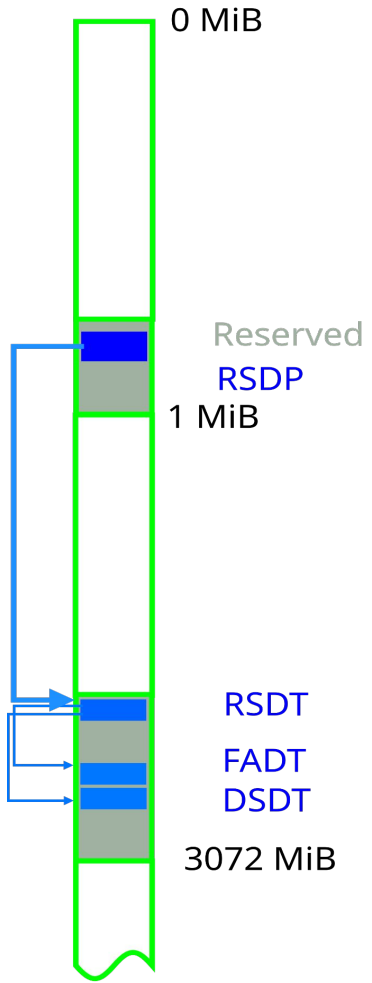
ACPI

- Replaces x86 legacy system software
  - BIOS interface is not standardized
  - BIOS calls are buggy
  - BIOS calls are slow
  - BIOS calls are 16-bit mode only
- Describes the hardware
- Lossless conversion between source and byte code
- Used by OS to implement generic drivers
- Configuration tables + AML byte code
- Requires AML interpreter in the OS
- Still depends on boot firmware
  - Loads/generates ACPI tables at boot
  - Installs tables in DRAM
  - Points OS to tables



ACPI

- RSDP
  - Root System Description Pointer
  - Points to RSDT
- RSDT
  - Root System Description Table
  - Pointers to other tables (FADT, IORT, MADT, …)
- FADT
  - Fixed ACPI Description Table
- DSDT
  - AML byte code
  - Tree like structured objects
  - each object has properties and methods
  - Needs an AML interpreter inside the OS
- SSDT
  - Like DSDT, but optional
  - Can have multiple SSDTs, but only one DSDT



x86 legacy memory map

SBB - Required ACPI tables

ACPI table	Supported in U-Boot	Required	
RSDP	✅	✅	Root System Description Pointer
RSDT/XSDT	✅	✅	Root System Description Table
DSDT/SSDT	✅	✅	Differentiated System Description Table
FADT	❌ **x86 only**	✅	Fixed ACPI Description Table
MADT	❌ **x86 only**	✅	Multiple APIC Description Table
SPCR	❌ **x86 only**	✅	Serial Port Console Redirection Table
DBG2	❌ **x86 only**	✅	Debug Port Table 2
MCFG	❌ **x86 only**	✅	PCI Memory-mapped Configuration Space

SBB - Required ACPI tables #2

ACPI table	Status in U-Boot	Required	
GTDT	❌	✅	Generic Timer Descriptor Table
PPTT	❌	✅	Processor Properties Topology Table
IORT	❌	❌	IO Remapping Table

devictree vs ACPI - example

```
/dts-v1/;

/ {
soc {
     compatible = "simple-bus";
     #address-cells = <2>;
     #size-cells = <2>;
     ranges;

     uart0 {
       compatible = "arm,pl011";
       reg = /bits/ 64 <SBSA_UART_BASE_ADDR
                        SBSA_UART_LENGTH>;
       status = "okay";
     };
  };
};
```

```
DefinitionBlock ("Dsdt.aml", "DSDT", 2,
"U-Boot", "SBSAQEMU", 2) {

  Scope (_SB) {



   // UART PL011
   Device (COM0) {
   Name (_HID, "ARMH0011")
   Name (_UID, Zero)
   Name (_CRS, ResourceTemplate () {
   Memory32Fixed (ReadWrite,
                  SBSA_UART_BASE_ADDR,
                  SBSA_UART_LENGTH)
   })

   Method (_STA) {
     Return (0xF)
   }
  }
}
```

devictree vs ACPI - example

```
/dts-v1/;

/ {
soc {
     compatible = "simple-bus";
     #address-cells = <2>;
     #size-cells = <2>;
     ranges;

     uart0 {
       compatible = "arm,pl011";
       reg = /bits/ 64 <SBSA_UART_BASE_ADDR
                    SBSA_UART_LENGTH>;
       status = "okay";
     };
  };
};
```

```
DefinitionBlock ("Dsdt.aml", "DSDT", 2,
"U-Boot", "SBSAQEMU", 2) {

  Scope (_SB) {



    // UART PL011
    Device (COM0) {
    Name (_HID, "ARMH0011")
    Name (_UID, Zero)
    Name (_CRS, ResourceTemplate () {
    Memory32Fixed (ReadWrite,
                   SBSA_UART_BASE_ADDR,
                   SBSA_UART_LENGTH)
    })

    Method (_STA) {
      Return (0xF)
    }
  }
}
```

devictree vs ACPI - example

```
/dts-v1/;

/ {
soc {
     compatible = "simple-bus";
     #address-cells = <2>;
     #size-cells = <2>;
     ranges;

     uart0 {
       compatible = "arm,pl011";
       reg = /bits/ 64 <SBSA_UART_BASE_ADDR
                    SBSA_UART_LENGTH>;
       status = "okay";
     };
  };
};
```

```
DefinitionBlock ("Dsdt.aml", "DSDT", 2,
"U-Boot", "SBSAQEMU", 2) {

  Scope (_SB) {



    // UART PL011
    Device (COM0) {
    Name (_HID, "ARMH0011")
    Name (_UID, Zero)
    Name (_CRS, ResourceTemplate () {
    Memory32Fixed (ReadWrite,
                   SBSA_UART_BASE_ADDR,
                   SBSA_UART_LENGTH)
    })

    Method (_STA) {
      Return (0xF)
    }
  }
}
```

devictree vs ACPI - example

```
/dts-v1/;

/ {
soc {
     compatible = "simple-bus";
     #address-cells = <2>;
     #size-cells = <2>;
     ranges;

     uart0 {
       compatible = "arm,pl011";
       reg =/bits/ 64 <SBSA_UART_BASE_ADDR
                       SBSA_UART_LENGTH>;
       status = "okay";
     };
  };
};
```

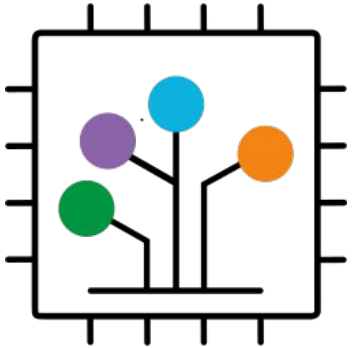
```
DefinitionBlock ("Dsdt.aml", "DSDT", 2,
"U-Boot", "SBSAQEMU", 2) {

  Scope (_SB) {



    // UART PL011
    Device (COM0) {
    Name (_HID, "ARMH0011")
    Name (_UID, Zero)
    Name (_CRS, ResourceTemplate () {
    Memory32Fixed (ReadWrite,
                   SBSA_UART_BASE_ADDR,
                   SBSA_UART_LENGTH)
    })

    Method (_STA) {
      Return (0xF)
    }
  }
}
```

devictree vs ACPI - example

```
/dts-v1/;

/ {
soc {
     compatible = "simple-bus";
     #address-cells = <2>;
     #size-cells = <2>;
     ranges;

     uart0 {
       compatible = "arm,pl011";
       reg = /bits/ 64 <SBSA_UART_BASE_ADDR
                        SBSA_UART_LENGTH>;
       status = "okay";
     };
  };
};
```

```
DefinitionBlock ("Dsdt.aml", "DSDT", 2,
"U-Boot", "SBSAQEMU", 2) {

  Scope (_SB) {



   // UART PL011
   Device (COM0) {
   Name (_HID, "ARMH0011")
   Name (_UID, Zero)
   Name (_CRS, ResourceTemplate () {
   Memory32Fixed (ReadWrite,
                  SBSA_UART_BASE_ADDR,
                  SBSA_UART_LENGTH)
   })

    Method (_STA) {
      Return (0xF)
    }
  }
}
```

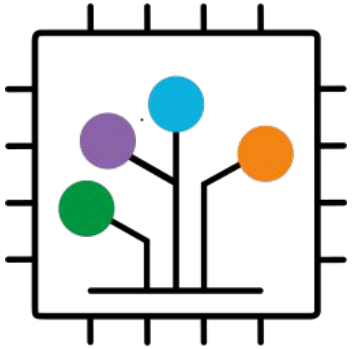
Devicetree - patching



- Flat devicetree inline patching
- Regenerate
  - Unpack (unflatten) to memory
  - Generate tree like structure with nodes
  - Modify tree
  - Pack tree (flatten) to new destination in memory

```
fdt_setprop_inplace_var(fdt, node, "linux,uefi-mmap-size", fdt_val32);
```

AML - patching



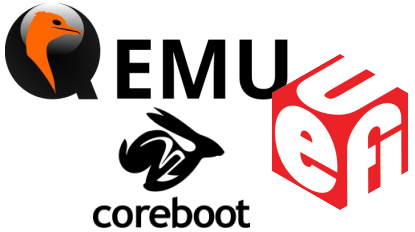
- Regenerate not supported
- Generating AML is not supported by EDK2
  - Inline patching
- Generating AML is supported by coreboot/U-Boot
  - C code generates AML byte code at runtime

```
if ((DsdtPointer[0] == 'A') && (DsdtPointer[1] == 'P') &&
    (DsdtPointer[2] == 'T') && (DsdtPointer[3] == '0' + Socket))
{
    DsdtPointer[9] = (UINT8) (mApicIdMap[Socket]);
}
```

Competitors

Before you start, learn from the best:

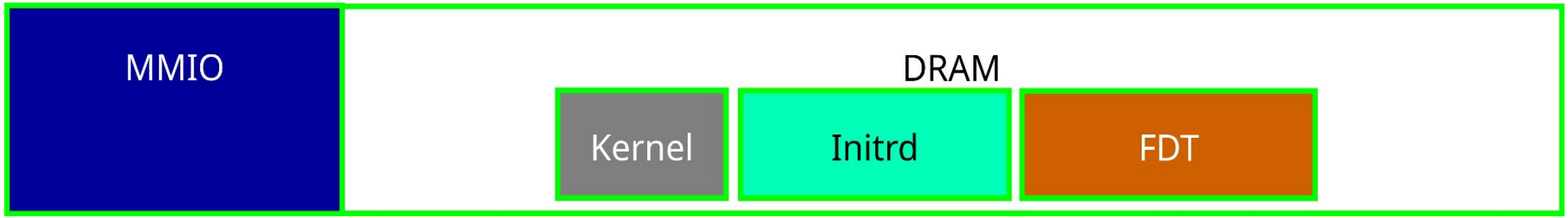
- Various existing solutions:
  - QEMU using FWCFG
  - UEFI/EDK2
  - coreboot



Competitors - QEMU

- Runtime generated ACPI tables before VM starts
- No firmware:
  - ACPI/devicetree is placed in DRAM
  - Kernel is placed in DRAM
  - Runs the kernel directly

```
- Primary CPU general-purpose register settings
  x0 = physical address of device tree blob (dtb) in system RAM.
  x1 = 0 (reserved for future use)
  x2 = 0 (reserved for future use)
  x3 = 0 (reserved for future use)
```

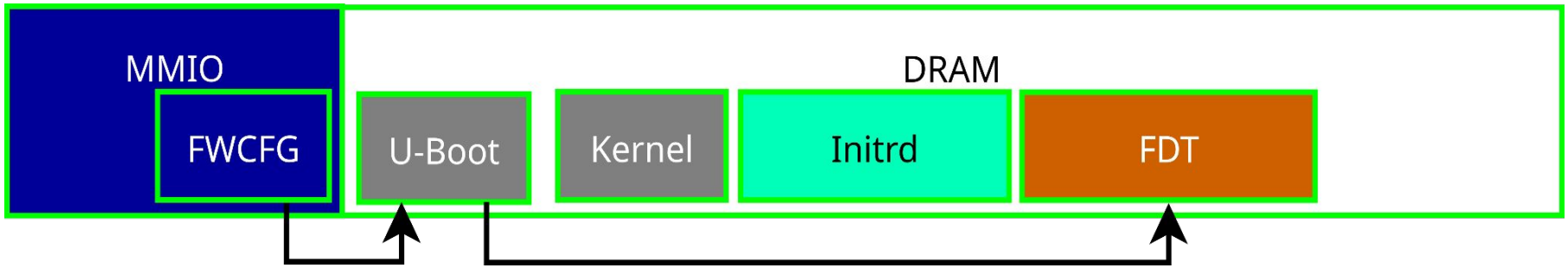


https://www.kernel.org/doc/Documentation/arm64/booting.rst

Competitors - QEMU

- Runtime generated ACPI tables before VM starts
- No firmware:
  - ACPI/devicetree is placed in DRAM
  - Kernel is placed in DRAM
  - Runs the kernel directly
- Firmware based: 

  - Firmware copies binary ACPI tables into VM using FWCFG
  - Firmware loads bootloader or kernel
  - Supports network boot and bootloader on disk



https://www.qemu.org/docs/master/specs/fw_cfg.html

Competitors - UEFI

- All major ACPI tables in flash
  - DSDT
  - SSDT
  - MCFG
  - DBG2
  - SPCR
  - MADT
  - PPTT
  - FADT
- Patched after loading to memory, even DSDT
- Some are runtime generated
- No devicetree used
  - uses defines in DSC

Competitors - coreboot

- Only DSDT in flash
- DSDT is not patched at runtime
- All other tables runtime generated by C code
- No devicetree used
  - Uses Kconfig
  - Mainboard specific code
  - uses runtime generated static.c (devicetree.cb)


```
acpigen_write_scope(scope); /* Scope */
acpigen_write_device(acpi_device_name(dev)); /* Device */
acpigen_write_name_string("_HID", cfg->hid);
acpigen_write_name_integer("_UID", cfg->uid);
acpigen_write_STA(acpi_device_status(dev));
acpigen_pop_len(); /* Device */
acpigen_pop_len(); /* Scope */
```

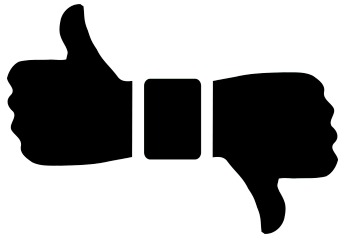
History / Acknowledgement

- Dec 15th 2021
  - Simon Glass
  - *“RFC: rpi: Enable ACPI booting on ARM with Raspberry Pi 4”*
- *June 19th 2024*
  - *Maximilian Brune*
  - *Rebased on upstream U-Boot*
  - *Moved x86 ACPI code into common folders*
  - *Call common ACPI code like x86 does*
- *July 30th 2024*
  - *Fixed unit-tests and linting*
  - *Send upstream for review*

History / Acknowledgement

- Dec 15th 2021
  - Simon Glass
  - *“RFC: rpi: Enable ACPI booting on ARM with Raspberry Pi 4”*
- *June 19th 2024*
  - *Maximilian Brune*
  - *Rebased on upstream U-Boot*
  - *Moved x86 ACPI code into common folders*
  - *Call common ACPI code like x86 does*
- *July 30th 2024*
  - *Fixed unit-tests and linting*
  - *Send upstream for review*

Requirements by U-Boot maintainers

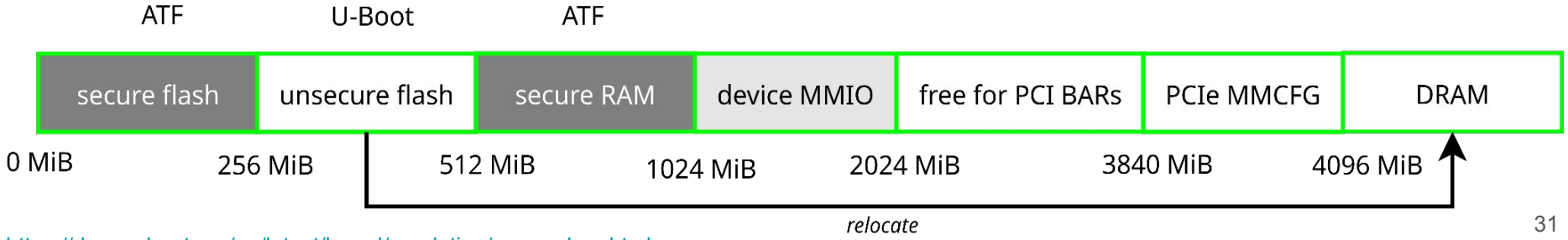


- DOs
  - “Emulated mainboard first”
    - Free and available to everybody
    - Allows others to easily confirm changes
    - Allows the CI to run the new code
  - “Devicetree first”
    - Every board MUST have a complete devicetree
    - Devicetree must be upstream to Linux first
    - No exceptions, even x86 boards have one
  - U-Boot drivers should take care of ACPI generation at runtime
    - U-Boot driver parse devicetree properties
    - U-Boot drivers install ACPI function PTRs
    - Generic ACPI code walks all drivers and thus writes out tables
- DON’Ts
  - No static DSDT AML code
  - No static ACPI tables in C

“Emulated mainboard first” - QEMU sbsa-ref

- Reference board for SBSA development
- QEMU generates a minimal DT
  - Number of CPUs
  - Amount of memory
  - HW version
  - Minimal GIC node
- QEMU doesn’t generate ACPI code
  - No FWCFG
- Need ATF as BL1/2
  - U-Boot runs as BL33

```
$ qemu-system-aarch64 -M sbsa-ref
```

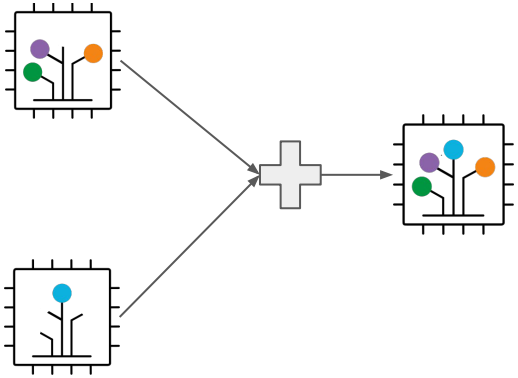


https://docs.u-boot.org/en/latest/board/emulation/qemu-sbsa.html

“Devicetree first” - QEMU sbsa-ref

- Devicetree provided by QEMU sbsa-ref is incomplete
  - U-Boot requires a devicetree at all costs
  - Usually devicetree must be in Linux upstream
  - Added proper devicetree in U-Boot
  - Included into U-Boot binary
- Merge QEMU provided devicetree at runtime

```
extern u8 __dtb_dt_begin[];     /* embedded device tree blob */

static inline u8 *dtb_dt_embedded(void)
{
        return __dtb_dt_begin;
}


static void *dtb_dt_qemu(void)
{
        /* FDT might be at start of DRAM */
        if (fdt_magic(SBSA_MEM_BASE_ADDR) == FDT_MAGIC)
                return (void *)(u64)SBSA_MEM_BASE_ADDR;
         return NULL;
}
```



“U-Boot drivers”

- U-Boot drivers match against “compatible”
  - Proper devicetree for every board required
- `ACPI_OPS_PTR` points to `struct acpi_ops`
- Invoked by core acpi function `acpi_get_method()`

```
static const struct udevice_id gic_v2_ids[] = {
	{ .compatible = "arm,gic-400" },
	{}
};

U_BOOT_DRIVER(arm_gic_v2) = {
	.name		= "gic-v2",
	.id		= UCLASS_IRQ,
	.of_match	= gic_v2_ids,
	ACPI_OPS_PTR(&gic_v2_acpi_ops)
};
```

“U-Boot drivers” - Example fill ACPI MADT

In the following example:

- U-Boot drivers fills `struct acpi_ops`
  - Implements `fill_madt`
- U-Boot drivers read in properties from the devicetree
- U-Boot drivers uses generic acpi writer
  - Here `acpi_write_madt_gicd()` is used to generate the structure

“U-Boot drivers” - Example fill ACPI MADT

```
static int acpi_gicv2_fill_madt(const struct udevice *dev, struct
acpi_ctx *ctx)
{
	struct acpi_madt_gicd *gicd;
	fdt_addr_t addr = dev_read_addr_index(dev, 0);
	if (addr == FDT_ADDR_T_NONE) {
		return -EINVAL;
	}

	gicd = ctx->current;
	acpi_write_madt_gicd(gicd, dev_seq(dev), addr, 2);
	acpi_inc(ctx, gicd->length);

	return 0;
}

static struct acpi_ops gic_v2_acpi_ops = {
	.fill_madt	= acpi_gicv2_fill_madt,
};
```

“U-Boot drivers” - Example fill ACPI MADT

```
static int acpi_gicv2_fill_madt(const struct udevice *dev, struct
acpi_ctx *ctx)
{
	struct acpi_madt_gicd *gicd;
	fdt_addr_t addr = dev_read_addr_index(dev, 0);
	if (addr == FDT_ADDR_T_NONE) {
		return -EINVAL;
	}

	gicd = ctx->current;
	acpi_write_madt_gicd(gicd, dev_seq(dev), addr, 2);
	acpi_inc(ctx, gicd->length);

	return 0;
}

static struct acpi_ops gic_v2_acpi_ops = {
	.fill_madt	= acpi_gicv2_fill_madt,
};
```

“U-Boot drivers” - Example fill ACPI MADT

```
static int acpi_gicv2_fill_madt(const struct udevice *dev, struct
acpi_ctx *ctx)
{
	struct acpi_madt_gicd *gicd;
	fdt_addr_t addr = dev_read_addr_index(dev, 0);
	if (addr == FDT_ADDR_T_NONE) {
		return -EINVAL;
	}

	gicd = ctx->current;
	acpi_write_madt_gicd(gicd, dev_seq(dev), addr, 2);
	acpi_inc(ctx, gicd->length);

	return 0;
}

static struct acpi_ops gic_v2_acpi_ops = {
	.fill_madt	= acpi_gicv2_fill_madt,
};
```

Generic code

- U-Boot’s generic code does all the heavy lifting
  - RSDT
  - RSDT/XSDT
  - DSDT
- Invokes U-Boot drivers for
  - MADT
  - DSDT
  - SSDT
  - NHLT
  - Generic table install
- Implemented in SoC/mainboard code (not using U-Boot drivers)
  - CSRT
  - FADT
  - IORT
  - PPTT
  - GTDT

Upstreaming status

- machines added
  - qemu-sbsa-ref (aarch64)
  - RPi4 ACPI (aarch64)
- Aarch64 ACPI support added
- Upstream completed in 2024
- Release tag v2025.01-rc2

```
113 files changed, 6519 insertions(+), 542 deletions(-)
```

ARM SBSA support on Rasberry Pi 4

Task: *Enable ARM SBSA on Raspberry Pi 4 within U-Boot*

- Basic ACPI support added
- Works on virtual machine:
  - qemu-system-aarch64 -machine raspi4b
- Boots on real hardware
- Not SBSA compliant
  - GiCv2 only (spec requires GiCv3)
  - No MCFG table
- Doesn’t implement quirks
- Mayor driver issues on vanilla linux
  - Lots of custom broadcom drivers designed around devicetree

→Please use the official EDK2 firmware for proper ACPI support!

Future plans

- Get rid of DSDT AML code
  - Everything is already part of devicetree
  - Should be converted at runtime from DT into ACPI
  - Add ACPI code generation in more U-Boot drivers
- Get rid of static tables in mainboard code
  - GTDT
  - PPTT
  - IORT
- Enable more boards